Credit Card Fraud Anomaly Detection and Prevention

John Klamut
University of Pittsburgh
jmk322@pitt.edu

Sushmit Acharya
University of Pittsburgh
sua26@pitt.edu

Aiden Seo
University of Pittsburgh
mis238@pitt.edu

Zeke Mariotti
University of Pittsburgh
eam180@pitt.edu

***Abstract*— With the pervasive use of credit cards in online and offline transactions, the risk of fraudulent activities has become a significant concern for financial institutions, businesses, and consumers alike. Credit Card Fraud detection is important in a variety of ways, and there are lots of methods of detection such as Rule-Based Systems or Machine Learning Models. We are going to implement anomaly detection to specifically utilize machine learning models pertaining to credit card fraud data. This involves identifying transactions that deviate from the "normal" behavior of the dataset. This may include large transactions, transactions in unusual locations, or transactions made at unusual times. We will be using a dataset from Kaggle.com which includes transaction information such as time and amount, as well as 28 features that were obtained through principal component analysis to prevent any of the users' personal information from being shown. The data is pre-processed to favor a machine learning based anomaly detection approach along with labeling for fraudulent transactions. We plan on evaluating multiple well-known machine learning models in order to evaluate their efficiency for our particular application.**

Keywords—Anomaly Detection, Credit Card Fraud, Machine Learning,

I. INTRODUCTION

Credit card fraud remains a challenge for financial industries in an era dominated by technology. Machine learning has emerged as a tool to fight credit card detection, offering multiple techniques to identify fraudulent activities [1].

Credit card detection implemented with machine learning recognizes patterns, anomalies, and trends within massive datasets. By training models on historical transaction data, these systems can discern subtle deviations indicative of fraudulent behavior, ranging from unusual spending patterns to irregular account activity. In this paper, we will explore different models of machine learning and their application in credit card detection. The four models that will be simulated are Support Vector Machine, Neural Network, XGBoost, and K-Nearest Neighbors. All the above models are supervised machine learning models that must be trained on a labeled dataset to accurately detect fraudulent data [1]. The data will be provided by Kaggle [2]. By using the same dataset, we can fairly compare the models. In the conclusion section, we will analyze the different models and compare the accuracies and true positive percentages.

II. CREDIT CARD DETECTIONG USING SUPPORT VECTOR MACINE

A. Model description

Support Vector Machine (SVM) is a supervised learning algorithm. Its objective is to find the optimal hyperplane that best separates data points belonging to different classes. SVM works by mapping input data into a dimensional space [3].

SVM can be effectively employed in credit card fraud detection tasks due to its ability to distinguish between legitimate and fraudulent transactions. In credit card detection, SVM works by utilizing historical transaction data to learn patterns and characteristics of both legitimate and fraudulent transactions. By training on labeled data (i.e., transactions labeled as either legitimate or fraudulent), SVM can identify complex patterns and boundaries that separate the two classes. SVM's capability to handle high-dimensional data and its ability to find optimal decision boundaries make it suitable for detecting fraudulent activities in credit card transactions. Once trained, the SVM model can classify new transactions as either legitimate or fraudulent based on the learned patterns [3]. Additionally, SVM can adapt to changing patterns of fraudulent activities because it is an anomaly-based detection system, making it a valuable tool in combating evolving fraud tactics.

The mathematical intuition and the following series of formulas are the basics of SVM and the simulation. First, we consider the equation for a hyperplane, $WX + B = 0$, where vector W is the direction perpendicular to the hyperplane. B is the offset or distance of the hyperplane from the origin in conjunction with W. Using this equation, points can classify the data point X into two groups: positive or negative based on whether they are below or above the hyperplane as seen in Figure 1 [4].

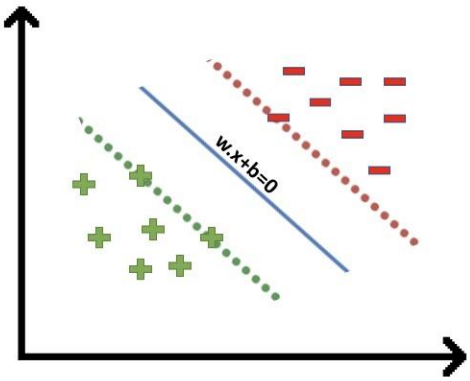

Figure 1: Visual Representation of SVM hyperplane with margins

At this point the SVM machine learning model can classify two different data groups based on one hyper plane. However, the current hyperplane is created based on two separate groups of data points. To achieve a more accurate hyperplane, as the one in Figure 1 shows, we need to find the margin from the hyperplane to the closest data point in both data groups. This margin is called the maximum margin. This point can be seen by the dashed green and red line in Figure 1. To find the maximum margin we can take two points classified as $(w.x + b) \geq 1$ and $(w.x + b) \leq -1$, and find the distance between them. The two points with the shortest distance are where we can accurately place our two hyperplanes. To find this distance we can use a trick with the dot product. With vector w, we can find the projection of the $(x2 - x1)$ as seen in Figure 2 [4].

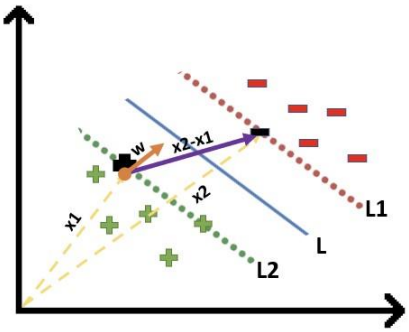

Figure 2: Finding the maximum margin to optimize SVM

Some other features of SVM include Hard and Soft margins. Soft margins add a tolerance to the equation which allows for outliers within the margin boundaries. In credit card detection, there is no tolerance for fraudulent transaction so the model will use a Hard margin which properly separated all data points. Additionally, SVM uses a kernel to optimize nonlinear datasets. In this case, the dataset can be properly separated on a 2D plane [5].

B. Simulation and results

The following simulation demonstrates the application of SVM. The simulation references the Sckit-learn libraries for machine learning algorithms [5]. Initially, the dataset is divided into two parts: one for training the machine learning model and the other for testing its performance. Since training the model with the entire dataset would be computationally demanding, only a portion of this dataset is utilized for training purposes. To train the model, the labeled data is plotted on a graph and the machine learning algorithm creates hyperplanes based on the equations described in the previous section. Once these planes are created, the SVM model will then be given a set of data for testing purposes that is unlabeled [6].

The SVM model will undergo training using two distinct datasets: one balanced and the other unbalanced. In the balanced dataset, an equal number of fraudulent and non-fraudulent transactions will be included, ensuring an even representation of both classes. Conversely, the unbalanced dataset will feature a random distribution of fraudulent and non-fraudulent instances, with one class significantly outnumbering the other. By training the SVM model on both balanced and unbalanced datasets, its performance and evaluation can be tested based on varying datasets, and we can further examine how SVM works in credit card detection.

The tables below show the true positive, false positive, true negative, and false negative results in the format of confusion matrixes for the balanced and unbalanced SVM models. Based on Figure 3, the SVM trained on balanced data was able to predict a fraud correctly 93% of the time with an accuracy of 86.53%. Compared to Figure 4 and the unbalanced dataset, the SVM model was only able to predict 62% of fraudulent data. However, because the unbalanced dataset was mostly composed of benign data points, accuracy was at 99.7% due to the limited about of fraudulent data points.

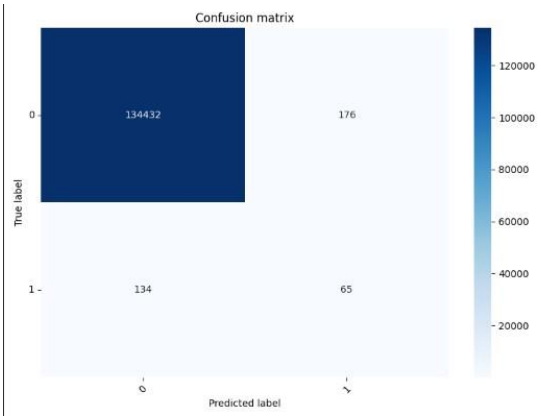

Figure 3: Confusion Matrix for SVM trained on Balanced Data

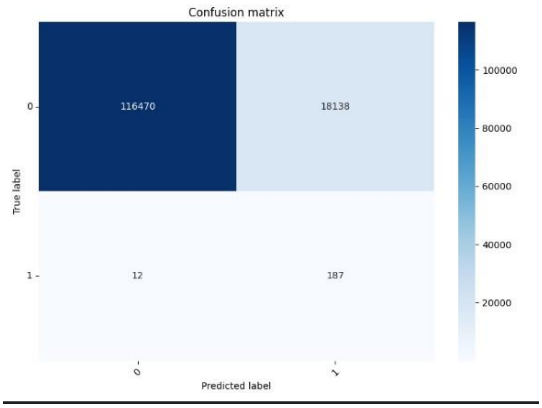

Figure 4: Confusion Matrix for SVM trained on Unbalanced Data

Based on the following results as seen in the table above, the model trained on the balanced data performed significantly better than the unbalanced data. In conclusion, the results are as expected. Due to how SVM separates data using a hyperplane, if there is a large unbalance in data, the resulting maxim margin between fraud and valid data can change greatly, which means there is a higher chance that the algorithm will classify fraud data as valid and valid data as fraud. Additionally with a large imbalance, the model won't have as many data points to accuracy create a hyperplane. This is why the percentage of correctly identified fraudulent for the unbalanced SVM is much lower than the balanced SVM.

III. Neural Networks

A. *Model description*

Neural Network is a supervised machine learning model inspired by the human brain's structure and functionality. The data path for a neural network-based machine learning model starts at the input layer where data is inserted in. After, the data goes through the hidden layers eventually delivering the output. Within each layer, we have neurons or activation units. These units compute the sum of the weighted inputs and then apply an activation function to that sum. Figure 5 shows a proper overview of the data path that starts from the input layer and ends at the output layer.

In terms of its applications towards the credit card fraud dataset, there are certain parameters and computations of interest. Before the supervised learning aspect, it is critical to standardize and normalize the data. After, the dataflow of the neural network model begins. This starts at the input layer which has every feature as a unit. Then we go through the designated number of hidden layers that use the RELU activation function.

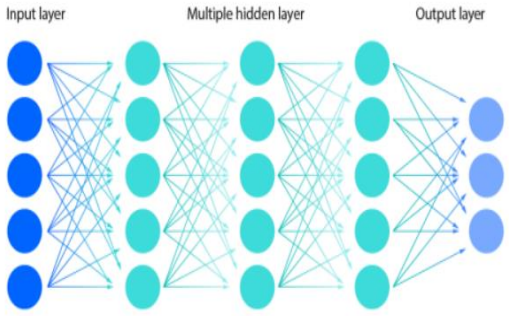

Figure 5: Neural Network Visual

B. *Simulation and results*

The following simulation results are from running the Neural Network model and analyzing its effectiveness. Some of the key decision making was derived from [7]. Figure 6 shows each of the layers and their shape. For the activation function each layer uses RELU while the output layer uses the sigmoid activation function.

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 16)	480
dense_1 (Dense)	(None, 24)	408
dropout (Dropout)	(None, 24)	0
dense_2 (Dense)	(None, 20)	500
dense_3 (Dense)	(None, 24)	504
dense_4 (Dense)	(None, 1)	25

Figure 6: Layer Description Table

Then, a key part of training and evaluating the accuracy of the model was dependent upon the epochs where an epoch refers to a complete iteration of the entire training data through the model's layers. Four different epoch values (1, 5, 10, and 20) were evaluated for the model, resulting in 5 epochs returning the highest accuracy (99.936%) amongst all the evaluated models.

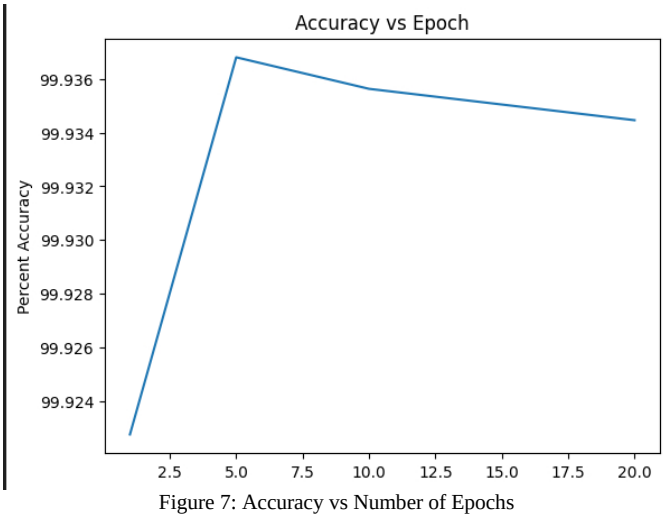

Figure 7: Accuracy vs Number of Epochs

Additionally, Figure 8 shows the confusion matrix that evaluated the model with 5 epochs. We see a high True Positive Rate of 99.80% with a False Positive Rate of 0.03% which is low as well.

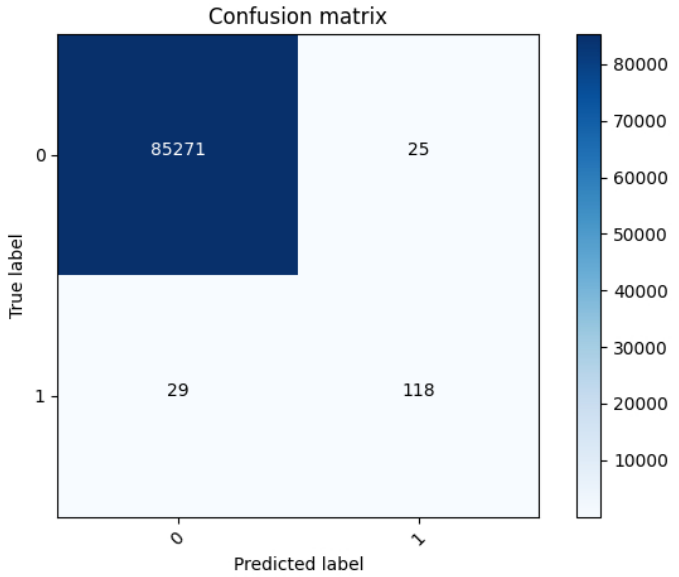

Figure 8: Confusion Matrix for Neural Network Model with 5 epochs

Overall, we see great performance with the Neural Network as expected due to the high-dimensional data and the neural network's ability to properly categorize such data.

IV. XGBoost

A. Model description

XGBoost or eXtreme Gradient Boosting is an efficient implementation of the gradient boosting algorithm. Gradient boosting is a learning algorithm that trains weak learners (i.e. simple models, typically shallow decision trees) with an emphasis on the parts of the data that have not been captured well so far. The final prediction of the model is a weighted average of the predictions of these weak learners. XGBoost has been repeatedly shown to be very effective in a wide variety of both regression and classification tasks.

XGBoost operates through a series of steps to iteratively filter its predictive capabilities. Initially, it initializes a model with a simple structure, usually a single leaf representing the mean value of the target variable. Then, it calculates the loss function, which measures the model's performance on the training data. By identifying optimal split points in the feature space, XGBoost constructs decision trees that minimize the loss function. These trees are added iteratively to the model, each one focusing on fitting the residual errors of the previous ones. Regularization techniques are applied throughout the process to prevent overfitting. XGBoost uses gradient boosting, adjusting the model based on the negative gradient of the loss function with respect to the predictions of the current model. Once training is complete, the model of trees is used to make predictions on new data. At the end, the model's performance is evaluated using appropriate metrics. This iterative approach allows XGBoost to build a robust model of decision trees capable of accurately capturing complex patterns in the data and making high-quality predictions [8].

B. Simulation and results

For the simulation, I used XGBClassifier class from the XGBoost package's Python API. This classifier implements Extreme Gradient boosting algorithm to optimize a given loss function. I used a logistic loss function, which is also the default loss of XGBoost for two-class classification tasks. To train the model, I used a credit card fraud detection dataset provided from Kaggle. To implement XGBoost algorithm for credit card fraud detection, I have referenced the code from a GitHub repository [9].

I first split the data into a training and a held-out test set. The test will only be used once at the very end of the model building process to provide an unbiased estimate of model performance on data it has never seen before.

Given the relatively large number of hyper-parameters of XGBoost, I used default XGBoost classifier model for hyper-parameter tuning to choose the set of hyper-parameters that achieve the highest cross validated Conditional Recall score. This is to provide control knobs to guide its learning. Then, I evaluated its performance on the credit card fraud detection dataset.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56864
1	0.96	0.78	0.86	98
accuracy			1.00	56962
macro avg	0.98	0.89	0.93	56962
weighted avg	1.00	1.00	1.00	56962

Figure 9: Prediction Table of the XGBoost simulation

Figure 9 shows the prediction Table of the XGBoost simulation. The recall rate of the 2nd row of the table is what represents the result of the simulation. As shown above, it gives us a 0.78 accuracy rate. In other words, it was able to detect 78% of the fraudulent credit card transactions correctly. Again, I used the XGBoost algorithm to detect fraudulent credit card transactions. The default XGBoost hyper-parameter optimization was used to efficiently tune the model as previously mentioned. Unfortunately, since credit card fraud detection must have at least 99% accuracy, it is evident from the result that XGBoost is not a powerful tool to use in this scenario.

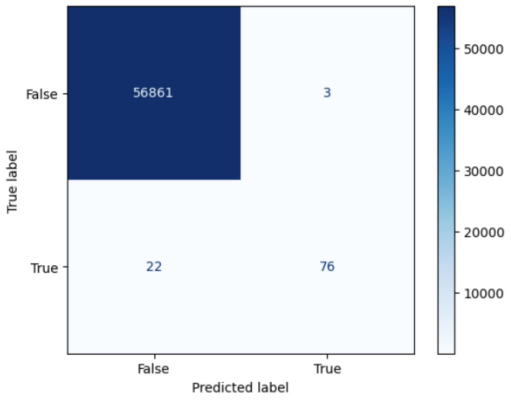

Figure 10: Confusion matrix for XGBoost

Figure 10 shows the confusion matrix of XGBoost, which is a tool used to evaluate the performance of a model and is visually represented as a table. According to the figure, the algorithm has 96% true positive rate with 78% accuracy rate, which is considered acceptable, but considering the accuracy rate, it is not a good result. Other measurements can also be observed from the confusion matrix such as true positive, false negative, and false positive rates. Overall, it is clear that XGBoost is not an appropriate algorithm for finding fraudulent credit card information from the dataset.

V. K-Nearest Neighbors

A. *Model description*

K-nearest neighbors (KNN) is a popular machine learning algorithm that relies on creating categories for a dataset and placing new data points into the group that is most similar. KNN is a supervised learning model that is most often used for classification applications. It is an algorithm that is known for being simple to implement while still being an effective solution for a wide range of machine learning problems [10].

The KNN algorithm starts with a value k, which decides the number of neighbors to chose for a given data point. Smaller k values are influenced more by outliers, while larger k values can result in a less meaningful separation of categories. When a new data point is entered into the algorithm, it finds the k number of neighbors with the shortest calculated distances to the new data point. Once the nearest neighbors are found, the category with the most neighbors is said to be the predicted class for the data point [10].

There are different methods for measuring the distance between points, with one of the most common being euclidean distance, which describes the distance of a straight line drawn between two points. Another type of distance calculation is called the manhattan distance, which calculates distance in a grid-like path, similar to navigating city blocks. The distance ends up being the sum of the different paths that are taken to travel from one point to the other [10].

Some of the advantages of the KNN algorithm include its simplicity of implementation compared to other machine learning models. KNN is a generalized algorithm, which makes it useful for many different types of datasets that may be structured in unusual ways. This also makes it useful for handling a wide variety of problems in machine learning [10].

One of the drawbacks of using KNN is that it starts to slow down significantly with larger datasets as well as taking up more resources. It can become less accurate if there are irelevant features used with the model. As well as this, the data must be normalized to be on the same scale so that the algorithm can operate properly [10].

B. *Simulation and results*

This section covers the application of KNN to identify credit card fraud within a dataset from Kaggle.com [2]. The model is based on an example created by Numan Yagoob with Python, and it mainly relies on the Scikit-Learn library to implement a KNN approach to detecting credit card fraud [10]. The Matplotlib and Seaborn libraries are used to visualize the results from running the model.

The first step of the simulation is to clean up the data by removing any transaction rows where the Class column has an invalid value. This is necessary since this column describes each transaction with a 1 if it was fraudulent and a 0 if it wasn't, and these values are needed to figure out if the prediction is correct. The data is then split into a training set and a test set, where the training set is used to train the model, and the test set is run with the model after it has been trained in order to create the prediction. After the model has been trained and the prediction is created with the test data, the results are visualized with a confusion matrix.

Fig. 11 shows the classification report for a model with k equal to 5. The precision value shows that 92% of the predictions that indicated a transaction as fraud were actually fraudulent. The recall value shows that 81% of the actually fraudulent transactions were predicted to be fraudulent by the model. The f1-score is a weighted harmonic average of precision and recall, and the support shows that among the testing dataset, there were 56,864 transactions that weren't fraudulent, and 98 transactions that were.

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56864
1	0.92	0.81	0.86	98
accuracy			1.00	56962
macro avg	0.96	0.90	0.93	56962
weighted avg	1.00	1.00	1.00	56962

Figure 11: Classification report for k=5

Fig. 12 shows the confusion matrix for the model when ran with k equal to 5. The vertical axis shows the actual values from the dataset, and the horizontal axis shows the predicted values from the dataset. The true negative rate was found to be 99.99%, with only 7 transactions being predicted as fraudulent when they weren't, which leaves a 0.01 false positive rate. The true negative rate came out to be 80.61%, meaning that 80.61%

of the transactions that were fraudulent were correctly predicted to be fraudulent by the model. This leaves the false negative rate at 19.39%.

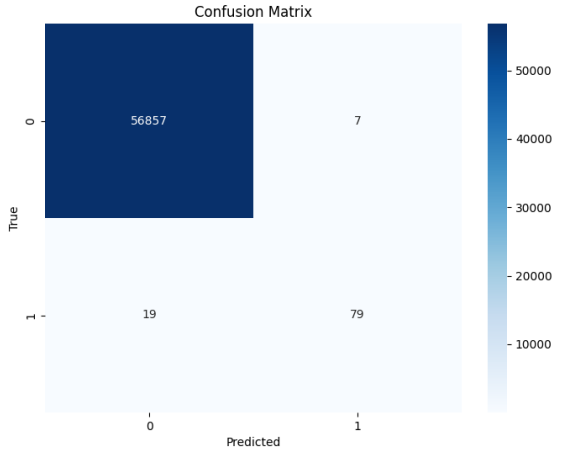

Figure 12: Confusion matrix for k=5

The results from the simulation show that KNN is a simple model with an easy implementation, but it is not perfect. A true negative rate of about 80% would not be ideal in a system that is designed to detect fraud, especially when there are millions of transactions being considered. Despite this, the model performed very well at correctly identifying non-fraudulent transactions with 99.99% accuracy. Although this model isn't able to detect fraud with incredible accuracy, it can still detect most fraudulent transactions, which would still make it a useful security measure for a system that oversees credit card activity.

VI. Conclusion

Overall, most of the models showed acceptable accuracy with Neural Networks being the most accurate as per [12]. Based on the results from all the simulations, Neural Networks had the highest accuracy at 99.93%. In terms of the other models, SVM had an accuracy of 93%, KNN had a 92% accuracy and XGboost had 78%. This was an expected outcome as they are known for properly predicting datasets with high dimensions in comparison to other models.

References

[1] Chupryna, Roman, and Olena Kovalenko. "Credit Card Fraud Detection Using Machine Learning - SPD Technology." Software Product Development Company, 27 Apr. 2021, spd.tech/machine-learning/credit-card-fraud-detection/.

[2] Machine Learning Group. "Credit Card Fraud Detection." Kaggle.com. https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

[3] "Support Vector Machine (SVM) Algorithm." GeeksforGeeks, GeeksforGeeks, 10 June 2023, www.geeksforgeeks.org/support-vector-machine-algorithm/.

[4] Saini, Anshul. "Guide on Support Vector Machine (SVM) Algorithm." Analytics Vidhya, 23 Jan. 2024, www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/.

[5] "1.4. Support Vector Machines." Scikit, scikit-learn.org/stable/modules/svm.html. Accessed 11 Apr. 2024.

[6] Sriram, Nuthi. "Credit Card Fraud Detection Using SVM." Kaggle, Kaggle, 10 Nov. 2019, www.kaggle.com/code/nuthisriram/credit-card-fraud-detection-using-svm/notebook.

[7] Bansal, Harsh. "Credit Card Fraud Detection." Analytics Vidhya, 28 Mar. 2021, medium.com/analytics-vidhya/credit-card-fraud-detection-c66d1399c0b7.

[8] Jonathan Kwaku Afriyie, Kassim Tawiah, Wilhemina Adoma Pels, Sandra Addai-Henne, Harriet Achiaa Dwamena, Emmanuel Odame Owiredu, Samuel Amening Ayeh, John Eshun, A supervised machine learning algorithm for detecting and predicting fraud in credit card transactions, Decision Analytics Journal, Volume 6, 2023, 100163, ISSN 2772-6622, https://doi.org/10.1016/j.dajour.2023.100163. (https://www.sciencedirect.com/science/article/pii/S2772662223000036)

[9] "Introduction to XGBoost in Python." Quantitative Finance & Algo Trading Blog by QuantInsti, 13 Feb. 2020, blog.quantinsti.com/xgboost-python/.

[10] Yagoob, Numan. "K-nearest neighbors algorithm for credit card fraud detection." PyFi. https://pyfi.com/blogs/articles/k-nearest-neighbors-algorithm-for-credit-card-fraud-detection

[11] RB, Asha, and Suresh Kumar KR. "Credit Card Fraud Detection Using Artificial Neural Network." Global Transitions Proceedings, vol. 2, no. 1, Jan. 2021, https://doi.org/10.1016/j.gltp.2021.01.006.